

## ETAPE 1: Jee WebApp, CRUD / BDD

### *Objectif*

Arriver à monter une application Web Jee comprenant:

- un formulaire
- agissant sur une unique table de BDD (table "Abonnés", par exemple: 4 champs Prénom, Nom, Âge, et une clé primaire) . Une seule table dans la BDD.
- Le formulaire doit permettre les opérations CRUD sur les enregistrements de la table de la BDD.

BOM:

- Tomcat,
- mariaDB,
- client HeidiSQL <https://www.heidisql.com/> (client graphique SQL)
- accès "management" BDD avec le user "lauriane/lauriane"
- accès BDD par l'application Web Java Jee avec l'utilisateur:
  - username: "appli-de-lauriane"
  - mot de passe: "mdp@ppli-1@urian3"

Autres installations (hors VM) :

- eclipse sur mon pc/mac habituel ("hôte" de virt.).
- HeidiSQL

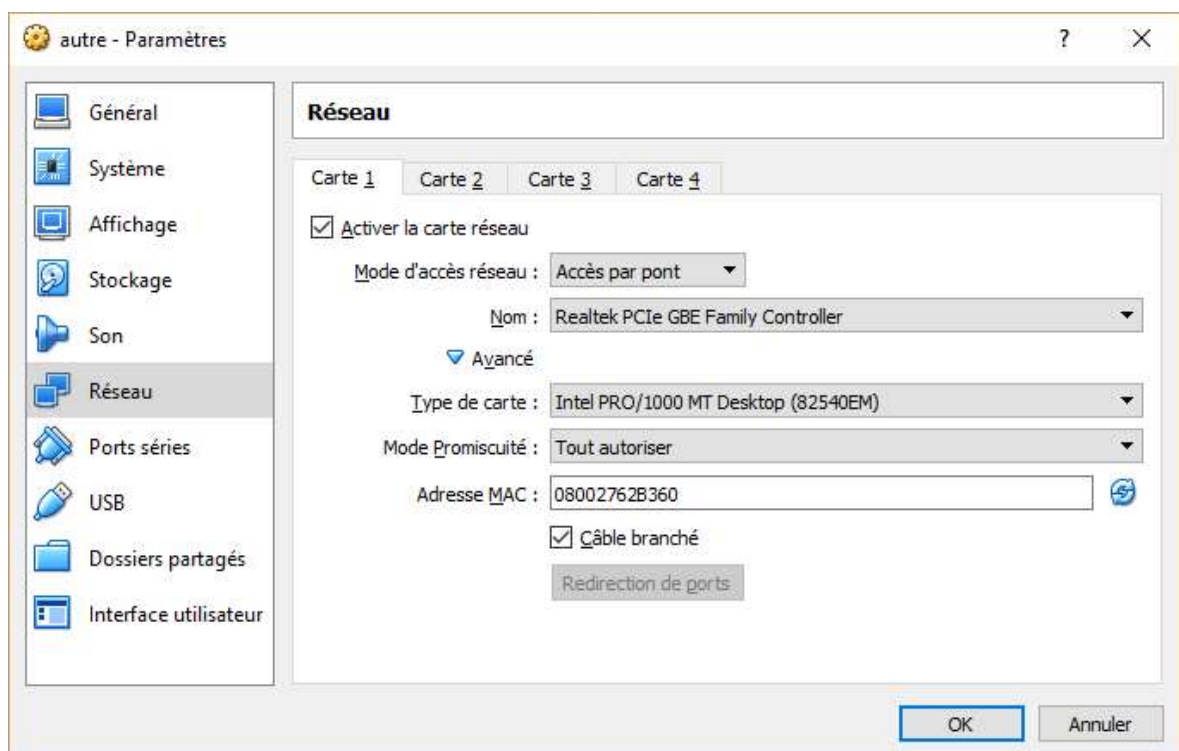
## Mode d'emploi:

- Nous allons créer une nouvelle VM Ubuntu. Pour pouvoir réaliser l'ensemble des opérations suivantes, la VM Ubuntu que tu vas créer:
  - doit avoir accès à Internet.

*(ou à un réseau IP, et dans ce réseau pouvoir accéder par le protocole TCP/IP, à ce que l'on appelle des "repository" ("dépôt") valides pour le logiciel "apt-get", mais ceci est une autre histoire).*

Un test simple: si dans ta VM Ubuntu, tu as accès à internet en ouvrant firefox, et que tu connais l'adresse IP de ta "box" FAI, alors tu as ce qu'il faut.

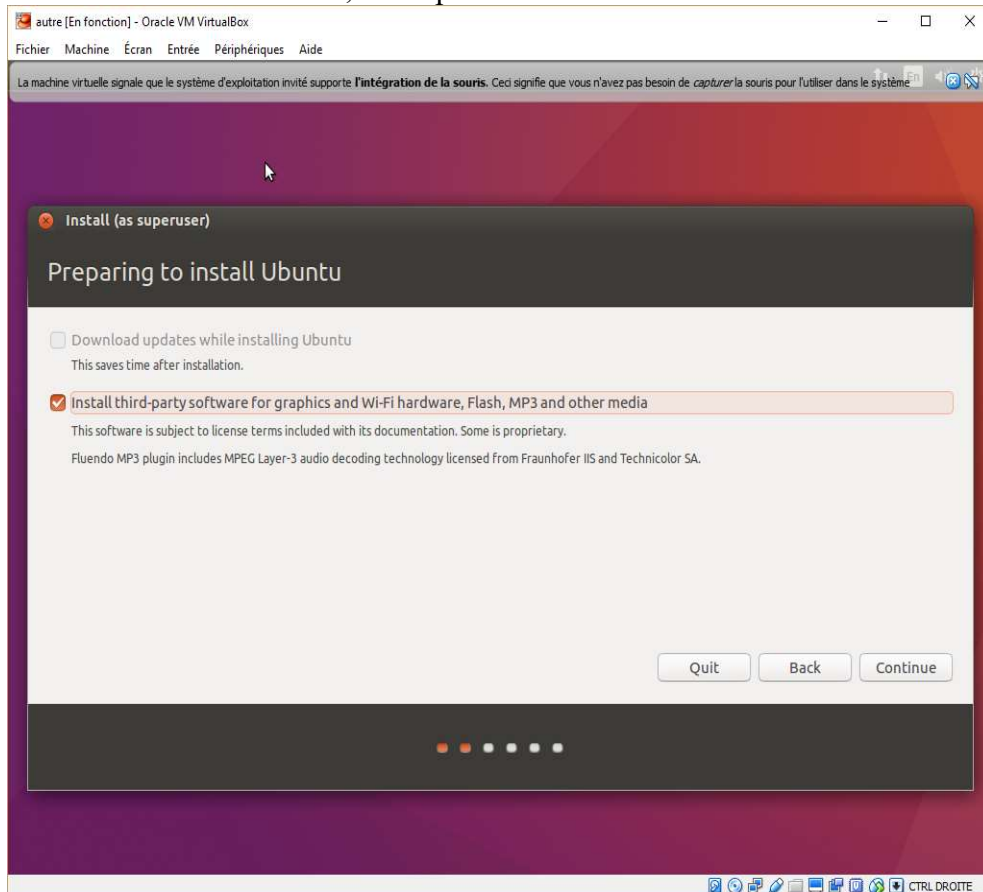
- Crées une nouvelle VM: tu y installeras donc Ubuntu. La carte réseau virtuelle de la VM doit être configurée :
  - avec un **Mode d'accès réseau** de type "Bridge Network" ("Accès par pont"),
  - et un **Mode Promiscuité** de type "Allow all" ("Tout autoriser"):



*Ci-dessus, on voit que j'ai configuré le mode d'accès réseau "Accès par pont", et le mode promiscuité "Tout autoriser" pour l'une des cartes réseau virtuelles, d'une VM Virtual Box que j'ai créée*

- Tu devra installer une version bien précise d'Ubuntu, la version "16.04 LTS". Tu peux télécharger une image ISO d'installation de cette version à l'aide du lien suivant (copies-colle ce lien dans la barre d'adresse Firefox):  
<ftp://ftp.free.fr/mirrors/ftp.ubuntu.com/releases/16.04/ubuntu-16.04.3-desktop-amd64.iso>
- Si avec le PC/MAC avec lequel tu travailles, tu es connectée à internet via wifi, alors,

pendant l'installation d'Ubuntu, à l'étape suivante de l'installation:



Tu devras cocher l'option "Install third-party software for graphics and Wifi hardware [...]": lorsque cette option est cochée, un pilote ("driver") de carte WIFI sera installé, ce qui permettra à ta VM de se connecter en Wifi.

À une étape ultérieure de l'installation, tu devras créer un utilisateur linux. Tu choisiras pour cet utilisateur le nom "lauriane-lx-usr" et le mot de passe "lemdp".

- Effectue maintenant l'installation Ubuntu.
- Quand l'installation est terminée, si ton PC/MAC est connecté en WIFI à internet, il faut configurer la connexion Wifi de ta nouvelle VM. Pour t'aider dans cette configuration, reportes-toi à l' Annexe I. "Configuration WIFI de la VM".
- Lorsque l'installation Ubuntu est terminée, et que tu as accès à Internet dans ta VM, ouvres un terminal, et exécutes les commandes:

```
sudo apt-get install -y git
git clone https://github.com/Jean-Baptiste-Lasselle/lauriane
# optionnellement, pour une version spécifique
# (exemple: la release "v2.0")
# git checkout tags/v2.0
# accessoirement, il serait logique de créer d'abord une branche:
# git branch mabranchedetravail
# pour ensuite faire le checkout dans la branche
# git checkout tags/v2.0 -b mabranchedetravail
# afin de pouvoir faire le merge plus tard...
```

- puis les commandes:

```
sudo chmod +x lauriane/monter-cible-deploiement.sh
sudo lauriane/monter-cible-deploiement.sh
```

- À cette étape, tu as un serveur Tomcat est prêt à être utilisé. Tu peux maintenant déployer une application Web Java Jee exemple. Pour cela ouvres un seond terminal (ou une session SSH avec PUTTY), et exécute la commande:

```
sudo lauriane/deployer-appli-web.sh
```

- pour déployer "un \*.war quelconque":

```
NOM_FICHER_WAR=./lauriane/nom-de-ton-fichier.war
sudo lauriane/deployer-appli-web.sh $NOM_FICHER_WAR
```

## ANNEXE III. Des commandes pour explorer déboguer

```
# sudo docker exec -it ciblededeploiement-composant-srv-jee
# sudo docker exec -it ciblededeploiement-composant-srv-jee /bin/bash -c "ls -all
/usr/local/tomcat/logs"
# sudo docker exec -it ciblededeploiement-composant-srv-jee /bin/bash -c "ls -all
/usr/local/tomcat/webapps"

# sudo docker exec -it ciblededeploiement-composant-srv-jee /bin/bash -c "cat
/usr/local/tomcat/logs/grep jdbc"

# sudo docker exec -it ciblededeploiement-composant-srv-jee /bin/bash -c "rm -rf
/usr/local/tomcat/webapps/jib1*"
# sudo docker exec -it ciblededeploiement-composant-srv-jee /bin/bash -c "rm -f
/usr/local/tomcat/logs/*"

# sudo docker exec -it ciblededeploiement-composant-srv-jee /bin/bash -c "cat
/usr/local/tomcat/bin/setenv.sh"

# sudo docker exec -it ciblededeploiement-composant-srv-jee /bin/bash -c "echo \"export
CATALINA_OPTS=\\\"\\\"$CATALINA_OPTS
-Dorg.slf4j.simpleLogger.defaultLogLevel=DEBUG\\\"\\\" >> /usr/local/tomcat/bin/setenv.sh"

# sudo docker exec -it ciblededeploiement-composant-srv-jee /bin/bash -c "cat
/usr/local/tomcat/conf/server.xml|grep organisation"
```

## ANNEXE III. Ajout d'un routeur au lab, pour s'isoler des autres réseaux

*La topologie qui va être montée*

On doit avoir 4 réseaux et dans notre cas, les réseaux qui seront créés et utilisés seront les 3 suivants:

- [RESEAU\_USINE\_LOGICIELLE] le réseau qui contient l'usine logicielle,
- [RESEAU\_USINE\_LOGICIELLE] le réseau qui contient la cible de déploiement,
- [RESEAU\_CIBLE\_DEPLOIEMENT] le réseau qui contient les machines physiques qui vous appartiennent (celle que vous utilisez personnellement pour le développement)

Ainsi qu'un quatrième réseau, le réseau qui contient la machine physique routeur qui donne accès à internet.

Ce dernier réseau peut par exemple être le réseau de votre domicile, et le routeur physique est alors votre box de votre FAI (your ISP's router), et il peut y avoir d'autres machines de la maison sur ce routeur. Ce peut aussi être le réseau de votre entreprise par lequel on vous donne le moyen d'accès à

internet. J'appellerai ci-après RESEAU\_EXTERIEUR, ce réseau.

Dans le cas, traité ci-après, les réseaux concernés auront les adressages IP suivants:

- RESEAU\_EXTERIEUR 192.168.1.0/24
- RESEAU\_USINE\_LOGICIELLE 192.168.2.0/24
- RESEAU\_CIBLE\_DEPLOIEMENT 192.168.3.0/24
- RESEAU\_MACHINES\_PHYSIQUES 192.168.4.0/24

Il y aura 3 routeurs à créer dans 3 machines virtuelles, 1 machine virtuelle qui contiendra 3 conteneurs docker et 1 serveur DHCP dans chaque conteneur les 4 réseaux doivent être séparés au niveau matériel, (au sens des couches du modèle OSI, en effet, le broadcast des adresses MAC pour chercher un serveur DHCP commence dès la connexion matérielle initiée, donc à partir du moment où une liaison ç un certain niveau OSI est établie)

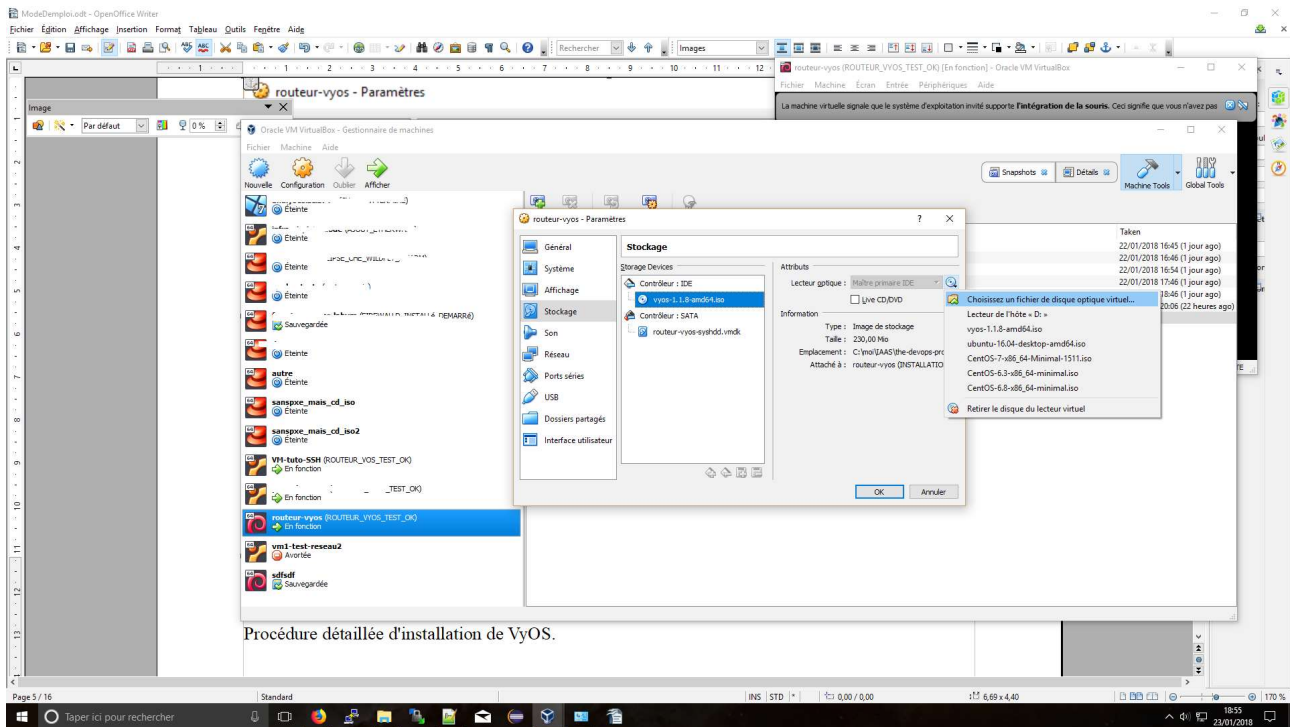
TODO: Là je donne des instructions pour faire l'ensemble du provisionning avec Virtual Box seulement, il faudra que 'étende à une utilisation pseudo open-stack de Virtualbox et KVM combinés, donc un petit inventaire des machines physiques enrôlées, puis configuration de la plateforme de virtualisation, qui sera utilisée pour le provisionning des VMs.

Dans mon cas personnel:

- RESEAU\_MACHINES\_PHYSIQUES contient un PC avec 16 Gb de RAM (mon pc de dev habituel, amd64), plus un serveur HP Proliant avec 50 Gb de RAM (sur celui-ci j'ai plus de place pour créer des VM)
- les autres réseaux contiennent uniquement des machines virtuelles, créées soit avec virtualbox (sur le pc dev 16Gb RAM), soit avec KVM (sur le serveur).

Procédure détaillée pas à pas installation/configuration de VyOS.

- Télécharger l'iso d'installation de VyOs à partir du site officiel: <https://vyos.io/>
- Sur Virtual Box, créer une VM:
  - ayant 4 cartes réseaux, toutes configurées en accès par pont, Mode promiscuité "Tout autoriser"
  - en fixant l'iso d'installation comme CD ROM configuré:



- Donnez 2048 Mo de RAM à la VM, et 2 processeurs
- démarrez la VM
- lorsque la séquence de boot se termine, un écran affiche un certain nombre d'informations, et le curseur est en attente, pressez simplement la touche entrée, aucune autre touche du clavier.
- Un login vous est demandé "vyos login: \_", saisissez "vyos" pour le login, et "vyos" pour le password qu'on vous demandera ensuite.
- Vous êtes loggué pour la première fois, et vyos n'est pas encore installé. Pour l'installer, vous devez taper la comande:

```
install image
```

- L'installation est lancée, et guidée. Laissez toutes les valeurs par défaut, à chaque fois qu'une valeur vous sera demandée.
- Voilà, VyOS est installé.
- Lorsque l'installation est terminée, on fait des constats:
  - en exécutant "ip addr" : permet de vérifier un point important : au contraire des distributions comme CentOS 7, ou ubuntu Server 14.04, les interfaces réseau linux configurées pendant le processus d'installation de VyOS ne sont PAS par défaut, configurées en DHCP. "ip addr" montre en effet, qu'elles n'ont pas d'adresses IP attribuées par ma "livebox", à laquelle les 4 cartes réseaux sont bien connectées puisqu'elles sont configurées pour un "Accès par pont". C'est une différence importante en "hôtes" et "routeurs", dans un réseau.
  - Il est possible d'exécuter des commandes sudo, avec l'utilisateur "vyos", et ce, sans que le mot de passe sudo soit demandé. Par exemple "sudo cat /etc/sudoers" est possible sans donner de mot de passe.
- Après ces constats, l'installation est terminée, et on va maintenant commencer à configurer le routeur VyOS, et exécuter des commandes spécifiques à VyOS (toutefois, la commande

"install image" utilisée précédemment, est une commande spécifique VyOS). Pour commencer, on a vérifié lorsque l'on a exécuté "ip addr", que 4 interfaces réseau linux ont été créées, une pour chaque carte réseau virtuelle (de la VM VirtualBox): "eth0", "eth1", "eth2", et "eth3". On a de plus constaté qu'aucune de ces interfaces réseau n'a d'adresse IP, elles ne sont configurées ni en dhcp, ni en ip statique.

- Pour que notre routeur puisse agir en tant que routeur pour un réseau **R**, il doit disposer d'une interface réseau linux configurée avec une adresse IP appartenant à **R**. Par exemple, pour que notre routeur puisse agir en tant que routeur pour le réseau "**RESEAU\_CIBLE\_DEPLOIEMENT**", il doit avoir une adresse IP dans "192.168.3.0/24", disons par exemple "192.168.3.17". L'habitude est à attribuer plutôt "192.168.3.1", mais pour l'expérience, je vous conseille de choisir "192.168.3.17". Bien, maintenant nous allons configurer l'interface réseau linux "eth2" de notre routeur, pour qu'elle ait l'adresse IP "192.168.3.17" dans le réseau "192.168.3.0/24" ("**RESEAU\_CIBLE\_DEPLOIEMENT**"), avec l'instruction suivante:

```
set interfaces ethernet eth2 address 192.168.3.17/24
commit
save
```

- Vérifiez, avec l'instruction "ip addr|grep 168", qu'une adresse IP est maintenant attribuée à "eth2". De plus, si vous créez rapidement 2 machines virtuelles Ubuntu Desktop, et leur appliquez la configuration IP statique ci dessous, vous pourrez vérifier que vous pouvez ping de l'une, vers l'autre, et réciproquement:

```
# et voilà, le routeur VyOS agit comme routeur sur le réseau 192.168.3.17
# Il suffit de créer 2 Vms dans ce réseau 192.168.3.0/24, pour voir
# si elles ont accès l'une à l'autre (ping)
# Disons 2 Vms Ubuntu de config à ajouter dans [/etc/network/interfaces] :
# -----
# VM1:
# auto enp0s3
# iface enp0s3 inet static
#     address    192.168.3.197
#     netmask     255.255.255.0
#     gateway     192.168.3.17
# -----
# VM2:
# auto enp0s3
# iface enp0s3 inet static
#     address    192.168.3.44
#     netmask     255.255.255.0
#     gateway     192.168.3.17
# -----
```

- Bien, procédez maintenant exactement de la même manière pour le réseau "**RESEAU\_USINE\_LOGICIELLE**", en prenant pour adresse IP du routeur dans ce réseau, l'adresse "192.168.2.1", et en utilisant l'interface réseau linux "eth1" du routeur. Vous vérifierez alors qu'une VM située dans "**RESEAU\_USINE\_LOGICIELLE**" peut envoyer un ping vers une VM du réseau "**RESEAU\_CIBLE\_DEPLOIEMENT**":

```

set interfaces ethernet eth1 address 192.168.2.1/24
commit
save

```

Et pour tester:

```

# le routeur VyOS agit comme routeur:
# - à l'adresse 192.168.3.17, pour le réseau 192.168.3.0/24,
# - à l'adresse 192.168.2.1, pour le réseau 192.168.2.0/24,
# Nous allons tester qu'une VM dans un des deux réseau, peut
# atteindre l'autre (ping).
# Il suffit de créer 2 Vms, une dans le réseau 192.168.3.0/24, une
# dans pour le réseau 192.168.2.0/24, puis de tenter les ping dans chaque.
# 2 Vms Ubuntu de config IP à ajouter dans [/etc/network/interfaces] :
# -----
# VM1:          (RESEAU RESEAU_CIBLE_DEPLOIEMENT 192.168.3.0/24)
# auto enp0s3
# iface enp0s3 inet static
#     address  192.168.3.127
#     netmask   255.255.255.0
#     gateway   192.168.3.17
# -----
# VM2:          (RESEAU RESEAU_USINE_LOGICIELLE 192.168.2.0/24)
# auto enp0s3
# iface enp0s3 inet static
#     address  192.168.2.51
#     netmask   255.255.255.0
#     gateway   192.168.2.1
# -----

```

- Dans votre réseau "RESEAU\_USINE\_LOGICIELLE", vous aurez typiquement des choses comme des serveurs jenkins, artifactory, et une Vm dans laquelle se trouve votre eclipse... Ils auront donc accès à la cible de déploiement. Il sera judicieux de voir qu'autoriser l'accès à l'usine logicielle, aux machines du réseau "RESEAU\_CIBLE\_DEPLOIEMENT", n'a pas de sens. Il sera donc judicieux d'affiner la configuration réseau pour qu'elle interdise ces accès. (Une possibilité pour ce faire est de mettre un second routeur et un serveur dhcp dans chaque réseau, on peut alors avec chaque routeur, pour chaque réseau, donner accès à 1 des 3 autres réseaux, simplement en configurant un interface supplémentaire en dhcp dans le réseau auquel on veut avoir accès, et en ajoutant la règle NAT appropriée)

```

#####
#####
# La topologie réseau cible
#####
#####
#
#

```



```

#
#
# Cette recette s'applique pour une VM ayant 4 cartes réseaux:
#
# - une en mode "Accès par pont": elle sera ainsi connectée au réseau physique dans
#   lequel se trouve la livebox, chez moi (la livebox contient un serveur DHCP, en
#   plus de jouer le rôle de routeur).
# - une en mode "Internal Network", et le nom du réseau interne VirtualBox, sera:
#   "RESEAU_USINE_LOGICIELLE" // routeur R1, cette VM
# - une en mode "Internal Network", et le nom du réseau interne VirtualBox, sera:
#   "RESEAU_CIBLE_DEPLOIEMENT" // sera en dhcp dans ce réseau, avec le routeur R2
# - une en mode "Internal Network", et le nom du réseau interne VirtualBox, sera:
#   "RESEAU_MACHINES_PHYSIQUES" // sera en dhcp dans ce réseau, avec le routeur R3
#
#
#
# Une fois un OS installé et configuré, cette VM sera le routeur R1, du réseau
# "RESEAU_USINE_LOGICIELLE", qui sera connecté via des interface réseau linux configurées
# en dhcp dans les autres réseaux:
#
#     ▫ avec le routeur R2 dans le réseau "RESEAU_CIBLE_DEPLOIEMENT"
#     ▫ avec le routeur R3 dans le réseau "RESEAU_MACHINES_PHYSIQUES"
#     ▫ avec le routeur de ma box internet dans le réseau "RESEAU_EXTERIEUR"
#
#
# Ainsi,
#
# Nous configurons tous les éléments des réseaux:
#     ▫ RESEAU_EXTERIEUR
#     ▫ RESEAU_USINE_LOGICIELLE
#     ▫ RESEAU_CIBLE_DEPLOIEMENT
#     ▫ RESEAU_MACHINES_PHYSIQUES
# sauf le routeur et le serveur DNS du réseau "RESEAU_EXTERIEUR"
#
#
# Soit N un des 4 réseaux:
#
#     ▫ RESEAU_EXTERIEUR
#     ▫ RESEAU_USINE_LOGICIELLE
#     ▫ RESEAU_CIBLE_DEPLOIEMENT
#     ▫ RESEAU_MACHINES_PHYSIQUES
#
#
# Tous les routeurs R1 à R3 auront la même configuration physique exactement (en tant
# que machines virtuelles), et ainsi, le routeur du réseau N:
#
#     - a une interface réseau linux par laquelle le routeur agit comme routeur dans
#       le réseau N,
#     - a 3 interfaces réseau linux configurées en dhcp dans les 3 autres réseaux.
#
# l'accès aux 3 autres réseaux, par une des machines du réseau N, ne pourra se faire que
# si une règle est ajoutée dans le routeur du réseau N, une règle NAT, par exemple.
# D'autre part, les 3 autres réseaux n'auront pas accès au réseau de l'usine
# logique, à moins que leur routeurs n'aient de règles configurés pour ce faire.
#
# Donc, avec ce montage, la liaison physique entre le routeur d'un des 2 réseaux:
#
#     ▫ RESEAU_CIBLE_DEPLOIEMENT
#     ▫ RESEAU_MACHINES_PHYSIQUES
#
# et le réseau "RESEAU_EXTERIEUR", n'existe pas.
#
# la liaison physique entre le routeur R1, du réseau "RESEAU_USINE_LOGICIELLE", et le
# réseau "RESEAU_EXTERIEUR" existe, car ce routeur a une carte réseau Virtuelle en
# "Accès par pont", à la machine physique
# (qui elle, se trouve dans le réseau du routeur FAI)
#
# la liaison physique entre les routeurs R2 et R3, et le "RESEAU_EXTERIEUR", peut
# être coupée, en ne créant que 3 cartes réseaux au lieu d'une, et aucune des 3
# cartes réseaux n'est physiquement connectée au "RESEAU_EXTERIEUR" (aucune n'est en
# "Accès par pont"). Les 3 cartes réseaux virtuelles sont alors configurées en

```

```

# "Internal Network", les réseaux internes choisis étant respectivement:
#
#     ▫ RESEAU_USINE_LOGICIELLE
#     ▫ RESEAU_CIBLE_DEPLOIEMENT
#     ▫ RESEAU_MACHINES_PHYSIQUES
#
# Remarque == >> Utiliser 3 "Internal network" virtual box, + un accès par pont à un
#     4 ième réseau, revient à se trouver dans la situation de 4 réseaux
#     PHYSIQUEMENT séparés.
#
#
# Ceci étant, dans ce cas d'utilisation d'un routeur par réseau, j'utilise 3 VMs rien
# que pour les routeurs, donc il est à voir comment faire plus efficace pour segmenter
# les réseaux.
# On pourrait: réduire le nombre de routeurs utilisés, en admettant que
# les machines du réseau de la cible de déploiement, puissent accéder aux
# machines du réseau de l'usine logicielle. On aurait alors seulement deux
# routeurs, histoire de bien garder la segmentation de sécurité autour
# des machines physiques. Seule l'usine logicielle peut accéder aux machines physiques.
#
# On aurait enfin une dernière option "utltra light": le routeur (2048 Mo RAM voire
# 1024 Mo, 2 vCPUs), la cible de déploiement en une seule VM (voire 2 maximum, soit
# 2 vCPUs et 4096 Go de RAM, éventuellement multiplié par 2), et
# une VM eclipse 6144 Mo de RAM 2 vCPUs.
#
# Dans tous les cas, cette dernière configuration ultra-light que je viens de donner, on
# peut faire rentrer ça sur un PC à 16 GO RAM intel i5 (2 coeurs, 4 Threads CPU).
#
#
# -----
# 4 cartes réseau au lieu de 3.
# -----
# "Internal Network" VBox | net id + netmask | interfaces réseau linux
# -----
# RESEAU_EXTERIEUR | 192.168.1.0/24 | eth0 (réseau de la livebox)
# RESEAU_USINE_LOGICIELLE | 192.168.2.0/24 | eth1 Le réseau de l'usine logicielle
# RESEAU_CIBLE_DEPLOIEMENT | 192.168.3.0/24 | eth2 Le réseau de la cible de déploiement
# RESEAU_MACHINES_PHYSIQUES | 192.168.4.0/24 | eth3 Le réseau des machines physiques
# -----
#
# -----
# **** Le réseau de l'usine logicielle:
# -----
#
# --- Sur pc de dev 16 Gb RAM (2 coeur AMD 64):
# - 1 [6 Gb RAM, 2 vCPUs] VM pour l'ide Eclipse,
# - 1 [8 Gb RAM, 2 vCPUs] VM pour 1 conteneur Artifactory, 1 conteneur gitlab,
#
# --- Sur serveur 50Gb RAM: (4 coeurs Xeon)
# - 1 [8 Gb RAM, 2 vCPUs] VM pour 1 autre conteneur gitlab, 1 conteneur Web Jee,
# - 1 [8 Gb RAM, 2 vCPUs] VM pour 1 autre conteneur Jenkins,
#
# Nota Bene: [1 conteneur Web Jee] ==>>> pour la gestion de la publication
# de la documentation (dans un site
# web projet interne) et autres publications
# vers l'extérieur comme les réseaux sociaux.
# -----
#
# -----
# **** Le réseau de la cible de déploiement:
# -----
#
# --- Sur serveur 50Gb RAM: (4 coeurs Xeon)
# - N * 3 VM [10 Gb RAM, 2 vCPUs] pour y jardiner des conteneurs dockers, des services
# kubernetes, ou des appliances openstack multi-tenant
#
#
# **** Le total des VMS sur serveur
# 4 coeurs Xeon, Redhat conseille moins de 10 vCPUs par
# coeur réel, et sur le serveur on est à 4 coeurs réels Xeon, avec
# 10 vCPUs créés en tout.

```

```

#
# On peut, au départ, réduire la taille de l'usine
# logicielle en la limitant à un des 2 VMs contenant un gitlab, et la VM IDE contenant.
# eclipse, ce qui fait 14 Go en tout. Il faut alors au
# moins 8 Go de RAM supplémentaire pour pouvoir monter une
# cible de déploiement un minimum utile.
# On va donc dire que notre environnement commence à être bien
# productif avec 22 Go de VM utilisables.

# **** Le réseau des machines physiques:
#
# le réseau dans lequel je mets toutes les machines que je
# veux particulièrement protéger, comme mes serveurs.
# C'est dans ce réseau qu'ils sont provisionnés en pixie.

#####
# 3./ On donne accès au réseau "RESEAU_CIBLE_DEPLOIEMENT",
# aux machines dans le réseau "RESEAU_USINE_LOGICIELLE":
# 192.168.2.0/24 [routeur: 192.168.2.1]
# 192.168.3.0/24 [routeur: 192.168.3.17]
#
#
#
# Sauf que cette fois-ci, il faut 2 routeurs:
#
# - routeur R1 pour le réseau "RESEAU_USINE_LOGICIELLE", et
# - routeur R2 pour le réseau "RESEAU_CIBLE_DEPLOIEMENT".
#
#####
# Pour cela, on va définir une règle NAT
# Tout ceci est test, et marche
#

#####
# MAINTENANT ON DEFINIT LA REGLE NAT
#####
# 1./ on entre en mode édition de la configuration
configure
# l'entier est le "numéro de règle (rule)": il est libre de choix (entre 1 et 9999).
edit nat source rule 12
# 2./ on définit l'interface "OUTBOUND" ==>> donc pour
# nous, l'interface réseau linux du routeur configuré en
# dhcp pour "RESEAU_CIBLE_DEPLOIEMENT", soit "eth2", c-à-d 192.168.3.0/24
set outbound-interface eth2
# 3./ on définit quelles adresses auront accès à l'interface
# "OUTBOUND" (définie juste avant, i.e. "eth2").
# Pour cette règle NAT, je spécifie un réseau entier, mais
# il est possible de spécifier d'autres
# manières, comme "tout sauf ce range d'adresses".
#
# Pour cette règle NAT, on donne donc accès à l'interface "OUTBOUND" (lié à
# "RESEAU_CIBLE_DEPLOIEMENT"), à toutes les VMs dans
# le réseau "RESEAU_USINE_LOGICIELLE" 192.168.2.0/24 :
set source address 192.168.2.0/24
# 4./ maintenant on définit la manière dont la
# "translation" (traduction) d'adresses se fait
set translation address masquerade

# enfin, on confirme la transaction, et on sauvegarde la configuration.
commit
save
# Maintenant:
# - les Vms dans le réseau "RESEAU_USINE_LOGICIELLE" peuvent faire un ping vers
# les Vms dans le réseau "RESEAU_CIBLE_DEPLOIEMENT".
# - les Vms dans le réseau "RESEAU_CIBLE_DEPLOIEMENT" ne peuvent PAS faire un ping vers
# les Vms dans le réseau "RESEAU_USINE_LOGICIELLE".

# Avec le même procédé, il est possible d'avoir un réseau de l'usine logicielle ayant

```

accès à tout les autres réseaux, les autres réseaux n'ayant pas accès au réseau de l'usine logicielle. Et pour chaque autre réseau, le réseau n'a accès à aucun autre réseau, pas même internet.

La page "", indique:

*"Internal Networking is similar to bridged networking in that the VM can directly communicate with the outside world. However, the "outside world" is limited to other VMs on the same host which connect to the same internal network.*

ccc

Donc en travaillant avec virtualbox, il serait plus affiné encore de modifier le montage précédent, pour que les cartes réseau virutelles virtualbox, correspondant aux interfaces réseau linux du routeur, connectées aux réseaux "RESEAU\_USINE\_LOGICIELLE", et "RESEAU\_USINE\_LOGICIELLE", soient configurées non pas en "Accès par pont", mais "Internal Network", si bien que les Vms des réseaux respectifs n'auront accès à internet que par "eth0", si on l'autorise. De plus les deux cartes réseaux peuvent être sur le même "Internal Network" ("Réseau interne"), ou sur deux différents. Dans ce cas, aura-ton des connectivités entre les deux réseau IP sur deux réseau "Internal Network" distincts? Normalement, elle sera possible grâce à configuration du routeur ? Les tests sont à faire

- 
- CCCC

cc

ccc

RESEAU\_EXTERIEUR

RESEAU\_USINE\_LOGICIELLE

RESEAU\_CIBLE\_DEPLOIEMENT

RESEAU\_MACHINES\_PHYSIQUES

Procédure détaillée d'installation de VyOS.

- On part de l'ISO
- 3 cartes réseaux sur la VM, toutes en accès par pont, Mode promiscuité "Tout autoriser"
- on lance la machine: on arrive à un écran où nous demande de presser la touche entrée pour commencer l'installation
- puis on se trouve directement à devoir s'identifier : on se connecte avec l'utilisateur/mdp "vyos/vyos", puis on exécute la commande d'installation :

```
install image
```

- l'installation est lancée, et guidée. Laisser toutes les valeurs par défaut.
- Lorsque l'installation est terminée, on fait des constats:
  - exécuter "ip addr" : permet de vérifier un point important : au contraire des distributions comme CentOS 7, ou ubuntu Server 14.04, les interfaces réseau linux configurées pendant le processus d'installation de VyOS ne sont PAS par défaut,

configurées en DHCP. "ip addr" montre en effet, qu'elles n'ont pas d'adresses IP attribuées par ma "livebox", à laquelle les 3 cartes réseaux sont bien connectées puisqu'elles sont configurées pour un "Accès par pont". C'est une différence importante en "hôtes" et "routeurs", dans un réseau, et on comprendra donc pourquoi au fil des opérations suivantes.

- Il est possible d'exécuter des commandes sudo, avec l'utilisateur "vyos", et ce, sans que le mot de passe sudo soit demandé. Par exemple "sudo cat /etc/sudoers" est possible sans donner de mot de passe.
- Après ces constats, l'installation est terminée, et on va maintenant commencer à configurer le routeur VyOS, et exécuter des commandes spécifiques à VyOS (toutefois, la commande "install image" utilisée précédemment, est une commande spécifique VyOS). Pour commencer, on a vérifié lorsque l'on a exécuté "ip addr", que 3 interfaces réseau Linux ont été créées, une pour chaque carte réseau virtuelle (de la VM VirtualBox): "eth0", "eth1", "eth2". On a de plus vérifié qu'aucune de ces interfaces n'a d'adresse IP. Nous allons mettre en œuvre la topologie suivante:
  - *Sans considérer le routeur que vous êtes en train d'installer en suivant ce document, vous avez déjà un réseau avec lequel vous travaillez, y compris si vous n'avez pas Internet: le réseau dans lequel se trouve la machine physique PC/MAC avec laquelle vous travaillez. La machine avec laquelle je travaille à la maison a l'adresse IP 192.168.1.14.*  
*Si votre machine n'a pas d'adresse IP, il est possible de lui en configurer une statique.*  
*En tout cas, la machine physique avec laquelle vous travaillez a une adresse IP, et appartient donc à un réseau IP. Chez moi, ma machine physique appartient au réseau 192.168.1.0/24. chez moi, c'est aussi le réseau sur lequel est accessible ma livebox, donc accès à Internet, avec l'adresse IP 192.168.1.1*  
*Ma machine physique appartient au réseau 192.168.1.0/24, et sur ce réseau, l'adresse IP du routeur qui donne accès à Internet est 192.168.1.1*
  - 3 réseaux connectés via ce routeur:
    - 192.168.1.0/24 ==>> le réseau dans lequel est ma machine physique. Dans ce réseau, c'est la livebox qui fait office de routeur pour tous les hôtes, y compris pour notre routeur VyOS.
    - 192.168.2.0/24 ==>> le réseau no. 1 dans lequel seront des VMs virtual box que je vais créer. Dans ce réseau, c'est notre routeur VyOS qui fera office de routeur pour tous les hôtes (toutes les VMs que l'on va créer), avec l'adresse IP 192.168.2.1
    - 192.168.3.0/24 ==>> le réseau no. 2 dans lequel seront des VMs virtual box que je vais créer. Dans ce réseau, c'est notre routeur VyOS qui fera office de routeur pour tous les hôtes (toutes les VMs que l'on va créer), avec l'adresse IP 192.168.3.1
    - Donc, si je PXE boot des VMs, il me faut configurer la config de isc-dhcp-server du pixie-node, pour que le serveur DHCP de la VM pixie bootée devienne celui de réseau no.2 ou no. 3... Vers un premier réseau unique, ce sera déjà un pas.
  - ccc
- Pour faire que 2 VMs dans le réseau no. 1 (192.168.2.0/24), puissent avoir accès l'une à l'autre, il faut et il suffit (sans qu'elles aient accès à Internet), qu'il y ait un routeur dans leur réseau. Ce routeur sera notre routeur VyOS, à l'adresse [192.168.2.23] . Pour que notre routeur VyOS agisse comme routeur dans ce réseau no. 2, et à l'adresse IP [192.168.2.23] , il nous faut configurer un des interfaces réseaux du routeur VyOS, pour qu'il ait une configuration IP avec {adresse= 192.168.2.23; netmask=255.255.255.0} :

```
set interfaces ethernet eth1 address 192.168.2.23/24
commit
save
# vérifiez, en exécutant la commande:
```

```
# ip addr|grep 168
```

```
# et voilà, le routeur VyOS agit comme routeur sur le réseau 192.168.2.23
```

```
# Il suffit de créer 2 Vms dans ce réseau 192.168.2.0/24, pour voir
```

```
# si elles ont accès l'une à l'autre (ping)
```

```
# Disons 2 Vms Ubuntu de config à ajouter dans [/etc/network/interfaces] :
```

```
# -----
```

```
# VM1:
```

```
# auto enp0s3
```

```
# iface enp0s3 inet static
```

```
#     address  192.168.2.37
```

```
#     netmask   255.255.255.0
```

```
#     gateway   192.168.2.23
```

```
# -----
```

```
# VM2:
```

```
# auto enp0s3
```

```
# iface enp0s3 inet static
```

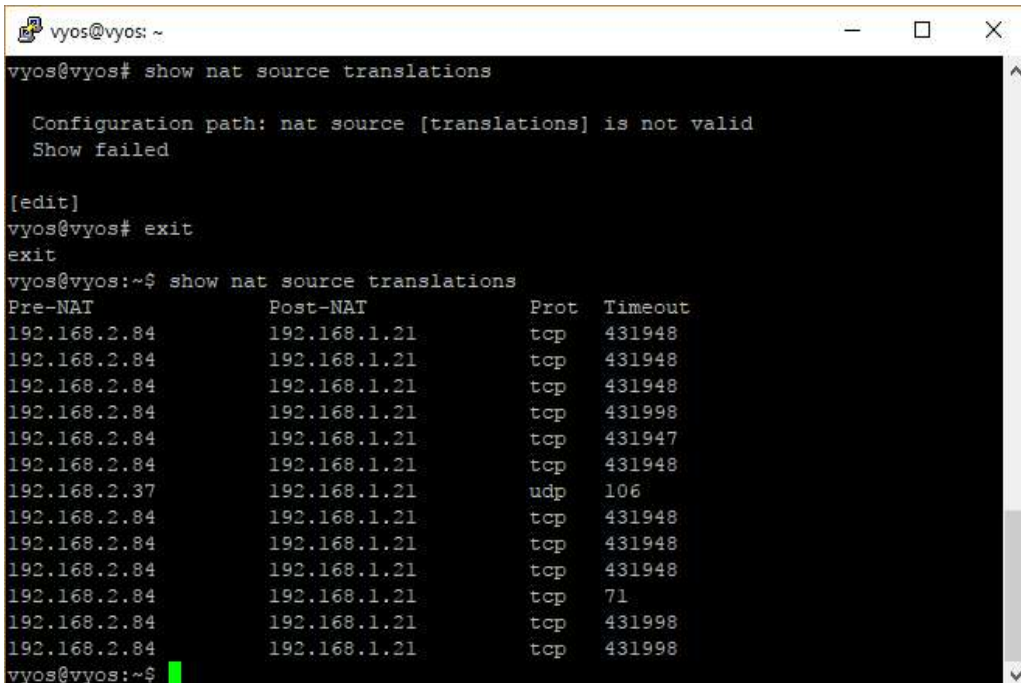
```
#     address  192.168.2.84
```

```
#     netmask   255.255.255.0
```

```
#     gateway   192.168.2.23
```

```
# -----
```

- En second temps, il faut faire en sorte qu'une VM dans le réseau no.1 ait accès au réseau internet. C'est bon, j'ai testé, et voilà le résultat du print de l'historique des translations d'adresses sur le routeur:



```
vyos@vyos# show nat source translations

Configuration path: nat source [translations] is not valid
Show failed

[edit]
vyos@vyos# exit
exit
vyos@vyos:~$ show nat source translations
Pre-NAT      Post-NAT      Prot  Timeout
192.168.2.84  192.168.1.21  tcp   431948
192.168.2.84  192.168.1.21  tcp   431948
192.168.2.84  192.168.1.21  tcp   431948
192.168.2.84  192.168.1.21  tcp   431998
192.168.2.84  192.168.1.21  tcp   431947
192.168.2.84  192.168.1.21  tcp   431948
192.168.2.37  192.168.1.21  udp    106
192.168.2.84  192.168.1.21  tcp   431948
192.168.2.84  192.168.1.21  tcp   431948
192.168.2.84  192.168.1.21  tcp   431948
192.168.2.84  192.168.1.21  tcp    71
192.168.2.84  192.168.1.21  tcp   431998
192.168.2.84  192.168.1.21  tcp   431998
vyos@vyos:~$
```

Ci-dessus, la trace de la traduction d'adresses ayant eu lieu en ouvrant firefox et en accédant à google.com, depuis la machine **192.168.2.84**

J'ai de plus testé que depuis ma machine physique d'adresse IP 192.168.1.15, je ne peux évidemment pas pinguer une VM du réseau no.1, mais par contre je peux pinguer ma machine

physique à partir d'une VM du réseau 1 (j'ai testé à partir de 192.168.2.84 vers 192.168.1.15).

- et la recette:

```
#####
# 3./ On donne accès internet, aux machines sur le réseau 1:
#      192.168.2.0/24 [routeur: 192.168.2.23]
#####
#
# Pour cela, on va définir une règle NAT
#
configure
#####
# MAINTENANT ON DEFINIT LA REGLE NAT
#####
# 1./ on entre en mode édition d'une règle NAT
#      l'entier qui est le "numéro de règle (rule)" est libre de choix.
edit nat source rule 12
# 2./ on définit l'interface "OUTBOUND" ==>> donc pour
#      nous "DEHORS", c-a-d 192.168.1.0/24, soit [eth0]
set outbound-interface eth0
# 3./ on définit quelles adresses auront accès à l'interface
#      "OUTBOUND" (défini juste avant)
#      ici, je spécifie un réseau entier, mais il est possible
#      de spécifier d'autres manières, comme "tout sauf ce range
#      d'adresses".
#      ici, on donne donc accès à l'interface "OUTBOUND", à
#      toutes les VMs dans le réseau 192.168.2.0/24
set source address 192.168.2.0/24
# 4./ maintenant on définit la manière dont la
#      "translation" (traduction) d'adresses se fait
set translation address masquerade

commit
save
```

- Dernier test à effectuer: automatiser le provisionning eclipse, dans une VM, dans le réseau no. 1, et dans le réseau no. 2, une autre VM avec le provisionning cible de déploiement, et vérifier que la VM eclipse a accès à la fois à la cible de déploiement, et à internet pour le repo github
- même chose que précédemment, sauf que je provisionne, en plus de la cible de déploiement, un gitlab privé, qui permet d'utiliser un repo git privé pour le repo assistant du deployeur-maven plugin.
- Dans les deux cas précédents, vérifier que l'on a pas accès à internet à partir d'une VM du réseau de l'infrastructure de déploiement, que l'on a depuis le réseau de la VM eclipse, accès à la fois au réseau de la cible de déploiement, et à internet.
- ccc

J'ai déjà testé que la configuration minimale pour faire communiquer 2 Vms en ip statique dans un réseau défini arbitrairement avec VyOS. Les 2 Vms n'ont pas accès à internet, et sont dans le réseau 192.168.2.0/24

La live box est dans le réseau 192.168.1.0/24

En l'état, 2 VMs et un routeur VyOS:

- VM ip statique: 192.168.2.37/24
- VM ip statique: 192.168.2.84/24
- routeur ip statique: 192.168.2.1/24

le test consiste à faire un ping d'une VM vers l'autre.

Ensuite je testerai si je peux créer 2 réseaux n'ayant ni l'un ni l'autre accès à internet, et ping d'un VM d'un réseau, vers une VM de l'autre réseau, grâce à une règle NAT que j'aurai configuré ( les VM du réseau 1 ont accès aux VM du réseau 2, mais pas l'inverse, et puis les 2 sens)

## ANNEXE II. Que la lumière soit

```
# =====>>> le grand principe <<< =====
# =====>>> le grand principe <<< =====

# ==>> donc 2 versionning:
#
#   == [VERSIONNING DE LA RECETTE] un pour le code de la recette de construction de la cible de déploiement
#   == [VERSIONNING DE LA DEPENDANCE] un repo git versionnant:
#
#   ++ fichier docker-compose.yml,
#
#   ++ les fichiers dockerfiles et fichiers permettant de construire (avec un docker build par exemple) l'image
#   customisée de chaque composant de l'infrastructure:
#
#   - [COMPOSANT SGBDR] (construire l'image mariadb avec un fichier de conf custom, permettant de changer le numéro de port):
#
#   le fichier dockerfile:
#
#       FROM mariadb
#       ADD ./mon.fichier.de.conf.custom
#       RUN cp ./mon.fichier.de.conf.custom /etc/mysql/my.cnf
#       CMD #là je crois c'est hérité du FROM
#
#   le fichier de conf custom mariadb: "./mon.fichier.de.conf.custom"
#
#
# - [COMPOSANT TOMCAT] (construire l'image mariadb avec un fichier de conf custom, permettant de changer le numéro de port)
#
#   et en fait, le code de la recette de déploiement effectue un checkout d'une version bien donnée du fichier docker-compose.yml
#   (et c'est là qu'est le lien entre le numéro de version de la recette, et le numéro de version de la dépendance que constitue le fichier
#   [docker-compose.yml])
#
# ==>> donc, en réalité, 1 repo git de versionning de la recette , et N repo git de chaque dépendance (de degré 1) de la recette.
#
# -----
# le lien aux principes "the devops program":
#
#   == une dépendance de degré zéro, cest: moi-même.
#   == une dépendance de degré zéro de la recette, cest le code source de la recette elle-même.
#
#   == une dépendance de degré zéro d'une appli java, c'est le code source de l'appli java elle-même.
#   == une autre dépendance de degré zéro d'une appli java, c'est le code source/config du build de l'appli java.
#
# -----
# !!!!!!!!!!!!!!! et des roues imbriquées, petite roue dans grande roue, il y a une dimension supplémentaire avec les multiples repos GIT:
# -----
# Donc, dans l'arbre des dépendances, pour une dépendance de degré N:
# - le versionning du code source de la dépendance de degré N, mentionne le lien aux versions des dépendances de degré N+1
# - le versionning du code source de la dépendance de degré N, permet de changer le numéro de version d'une dépendance de degré N+1
# !!!!!!!!!!! par exemple, pour changer le numéro de port de mariadb de la cible de déploiement
# !!!!!!!!!!! (par config de mariadb et / ou changement de mapping docker numéro de port du conteneur docker ) :
#
#   ## ON DETRUIT LA CIBLE DE DEPLOIEMENT
#   on fait docker-compose down
#   ## ON EDATE DU CODE SOURCE
#   on fait une nouvelle version du fichier "docker-compose.yml" en changeant dans ce fichier le mapping du
#   numéro de port dans le et/ou le numéro de version checkouté pour le checkout du repo versionnant le
#   dockerfile pour construire l'image (FROM mariadb ADD ./mon.fichier.de.conf.custom RUN cp ./mon.fichier.de.conf.custom /etc/mysql/my.cnf)
#   on édite sed le fichier "./docker-compose.yml", pour changer le mapping docker du numéro de port d'écoute de mariadb avec un numéro de
#   port exposé par le conteneur du composant SGBDR
#   on édite sed le fichier de conf custom mariadb "/mon.fichier.de.conf.custom", pour changer le numéro de
#   port d'écoute de mariadb ) "à l'intérieur du conteneur"
#   ## ON COMMIT l'un ou l'autre, ou les deux...
#   [git add "./docker-compose.yml"]
#   [git commit -m "nouveau mapping docker numéro de port mariadb -p noportexterne=noportinterne"]
#   [git add "./mon.fichier.de.conf.custom"]
#   [git commit -m "nouveau numéro de port mariadb noportexterne="]
#   ## ON TAGGUE
#   on détruit le tag ETAT_INITIAL_CIBLE_DEPLOIEMENT : [git tag -d vETAT_INITIAL_CIBLE_DEPLOIEMENT] (san détruire le commit associé au tag détruit)
#   on git taggue le commit [git tag -a vETAT_INITIAL_CIBLE_DEPLOIEMENT -m "ETAT_INITIAL_CIBLE_DEPLOIEMENT: nouveau numéro de port etc..."]
#   on fait un git cdocker-compose up
#   on fait docker-compose up
#   ## IL MANQUE LA PARTIE DOKER BUILDS AVEC DOCKER COMMIT POUR VERSIONNER IMAGES MONTÉES ...?
#
#
```



le grand principe: IL MANQUE LA PARTIE DOCKER BUILDS AVEC DOCKER COMMIT POUR VERSIONNER IMAGES MONTEES  
mais pour moi on pourrait ne pas changer les numeros de version

## ANNEXE I. Configuration WIFI de la VM

- Pour ce faire, tu as besoin:
  - Du nom de la connexion Wifi: tu dois le connaître, car pour connecter ton PC/MAC en Wifi, tu as du rechercher le "nom de la connexion Wifi". Exemple de nom de connexion Wifi: "Livebox-3818". Plus précisément, c'est l'identifiant du point d'accès Wifi que tu utilises.
  - Et si la connexion Wifi n'est pas libre, d'autres informations, par exemple un mot de passe (c'est souvent le cas dans des lieux ouverts au public comme les bar et bibliothèques). Je vais supposer qu'un mot de passe est suffisant pour être autorisé à se connecter en Wifi, et si ce n'est pas le cas....: débrouilles-toi pour connecter ta VM en gros comme tu l'as fait pour connecter ton PC / MAC.

Tu disposes donc de l'identifiant (le nom) du point d'accès Wifi, et du mot de passe. Pour configurer ton wifi:

1. exécutes la commande:

```
sudo gedit /etc/network/interfaces
```

2. Regardes le contenu de ce fichier, tu devrais trouver un bloc qui ressemble à ceci (ici, j'ai mis "wlan0, mais ce pourrait être "wlan1", "wlan2", "wlan3" ...):

```
auto wlan0
iface wlan0 inet dhcp
```

3. il y a peut-être deux lignes supplémentaires sous ce bloc (si elle n'y sont pas, ajoutes-les):

```
auto wlan0
iface wlan0 inet dhcp
wpa-ssid etquelquechoslà
wpa-psk etautrechoseici
```

4. édites le fichier pour insérer l'identifiant (le nom) du point d'accès Wifi, et le mot de passe comme ci-dessous:

```
auto wlan0
iface wlan0 inet dhcp
wpa-ssid 1-identifiant-du-point-d-acces-wifi
wpa-psk et-le-mot-de-passe-ici
```

5. Nota Bene: les accès WIFI libres existent encore à certains endroits, et dans ce cas, n'ajoutes pas la dernière ligne (précisant le mot de passe):

```
auto wlan0
iface wlan0 inet dhcp
wpa-ssid 1-identifiant-du-point-d-acces-wifi
```

6. Enregistres (Ctrl +S), quitte gedit (Ctrl + Q), puis exécutes dans un terminal:

```
ip addr flush wlan0
systemctl restart networking.service
```

Ta VM devrait maintenant avoir accès à internet. Si ce n'est pas le cas, alors.. Eh bien tu vas te débrouiller, mais voici quelque liens qui pourraient t'aider:

<https://doc.ubuntu-fr.org/wifi>

[https://doc.ubuntu-fr.org/wifi\\_ligne\\_de\\_commande](https://doc.ubuntu-fr.org/wifi_ligne_de_commande)

Bon courage! ;)

## ANNEXE III. Configuration d'un data source pour applications Jee

- **Pour Tomcat**

- Pour le cas datasource au niveau du serveur:
  - [ docker cp dans le conteneur ] vérifier la présence de `$CATALINA_HOME/lib/tomcat-dbcp.jar` , le déployer si nécessaire
  - [ docker cp dans le conteneur ] déployer dans `$CATALINA_HOME/lib` le jar du driver JDBC
  - [ docker cp dans le conteneur ] configuration à appliquer dans le `[$TOMCAT_HOME/conf/context.xml]` (à vérifier par META-INF/context.xml du projet):

vierge, ce fichier a pour contenu:

```
<?xml version='1.0' encoding='utf-8'?>
<!-- Licensed to the Apache Software Foundation (ASF) under one or more contributor
    license agreements. See the NOTICE file distributed with this work for additional
    information regarding copyright ownership. The ASF licenses this file to
    You under the Apache License, Version 2.0 (the "License"); you may not use
    this file except in compliance with the License. You may obtain a copy of
    the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required
    by applicable law or agreed to in writing, software distributed under the
    License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS
    OF ANY KIND, either express or implied. See the License for the specific
    language governing permissions and limitations under the License. -->
<!-- The contents of this file will be loaded for each web application -->
<Context>

    <!-- Default set of monitored resources. If one of these changes, the -->
    <!-- web application will be reloaded. -->
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
    <WatchedResource>${catalina.base}/conf/web.xml</WatchedResource>

    <!-- Uncomment this to disable session persistence across Tomcat restarts -->
    <!-- <Manager pathname="" /> -->

    <!-- Uncomment this to enable Comet connection tacking (provides events
        on session expiration as well as webapp lifecycle) -->
    <!-- <Valve className="org.apache.catalina.valves.CometConnectionManagerValve"
        /> -->
</Context>
```

et on y ajoute la balise `<Resource>`:

```
<?xml version='1.0' encoding='utf-8'?>
<!-- Licensed to the Apache Software Foundation (ASF) under one or more contributor
    license agreements. See the NOTICE file distributed with this work for additional
    information regarding copyright ownership. The ASF licenses this file to
    You under the Apache License, Version 2.0 (the "License"); you may not use
    this file except in compliance with the License. You may obtain a copy of
    the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required
    by applicable law or agreed to in writing, software distributed under the
    License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS
    OF ANY KIND, either express or implied. See the License for the specific
    language governing permissions and limitations under the License. -->
<!-- The contents of this file will be loaded for each web application -->
<Context>

    <!-- Default set of monitored resources. If one of these changes, the -->
    <!-- web application will be reloaded. -->
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
    <WatchedResource>${catalina.base}/conf/web.xml</WatchedResource>

    <!-- Uncomment this to disable session persistence across Tomcat restarts -->
    <!-- <Manager pathname="" /> -->

    <!-- Uncomment this to enable Comet connection tacking (provides events
        on session expiration as well as webapp lifecycle) -->
    <!-- <Valve className="org.apache.catalina.valves.CometConnectionManagerValve"
        /> -->
    <!-- Configuration du datasource -->

<Resource name="organisation/SourceDeDonnees" auth="Container" type="javax.sql.DataSource"
    maxActive="20"
    maxIdle="10"
    maxWait="10000"/>
```

```

        username="lauriane"
        password="lauriane"
        driverClassName="org.mariadb.jdbc.Driver"
        url="jdbc:mariadb://localhost:8456/bdd_oraganisation"
    />
</Context>

```

et pour finir [ docker restart du conteneur ]

[Donc il me faut comme paramètres supplémentaires du plugin]

Ci-dessous, une correspondance entre les balises d'un pom.xml utilisant le plugin, et les variables d'environnement utilisées dans les scripts de déploiement du datasource:

- MARIADB\_JDBC\_DRIVER\_CLASS\_NAME=  
`<jdbc-driver-classname>org.mariadb.jdbc.Driver</jdbc-driver-classname>`
- JEE\_DATASOURCE\_NAME=  
`<jee-datasource-name>organisation/SourceDeDonnees</jee-datasource-name>`
- JEE\_DATASOURCE\_AUTH\_USERNAME=`<lx-user>lauriane</lx-user>`
- JEE\_DATASOURCE\_AUTH\_USERPWD=`<lx-pwd>lauriane</lx-pwd>`
- JEE\_DATASOURCE\_MAX\_ACTIVE=  
`<jee-datasource-max-active>20</jee-datasource-max-active>`
- JEE\_DATASOURCE\_MAX\_IDLE=  
`<jee-datasource-max-idle>10</jee-datasource-max-idle>`
- JEE\_DATASOURCE\_MAX\_WAIT=  
`<jee-datasource-max-wait>10000</jee-datasource-max-wait>`
- JEE\_DATASOURCE\_URL\_PREFIX=  
`<!-- ici il y a le numéro de port et adresse IP SGBDR-->`  
`<jee-datasource-url-prefix>jdbc:mariadb</jee-datasource-url-prefix>`
- JEE\_DATASOURCE\_URL=  
`<!-- Non, inutile dans la configuration, l'URL peut être formée à partir:`  
  - de l'adresse IP utilisée par le SGBDR
  - du numéro de port utilisé par le SGBDR
  - de la valeur de \$JEE\_DATASOURCE\_URL\_PREFIX `<jdbc-datasource-url-prefix>`
  - de la valeur de \$NOM\_BDD\_APPLI `<jdbc-datasource-url-prefix>``-->`  
`<jee-datasource-url>jdbc:mariadb://192.168.1.149:3306/db</jee-datasource-url>`
- dans [WEB-INF/web.xml], ajouter une référence au datasource configuré dans le [\$TOMCAT\_HOME/conf/context.xml] ccc

```

<resource-ref>
  <description>DB Connection</description>
  <res-ref-name>jdbc/TestDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>

```

- Enfin, pour tester le datasource, il faut:
  - créer une table de tests "MembresAssos" dans la BDD, et y mettre quelques enregistrements. Exemple de table:

MembresAssos  
 prenom

nom  
username  
email  
age

- Pour le cas datasource au niveau de l'application:

// à préciser, mais:

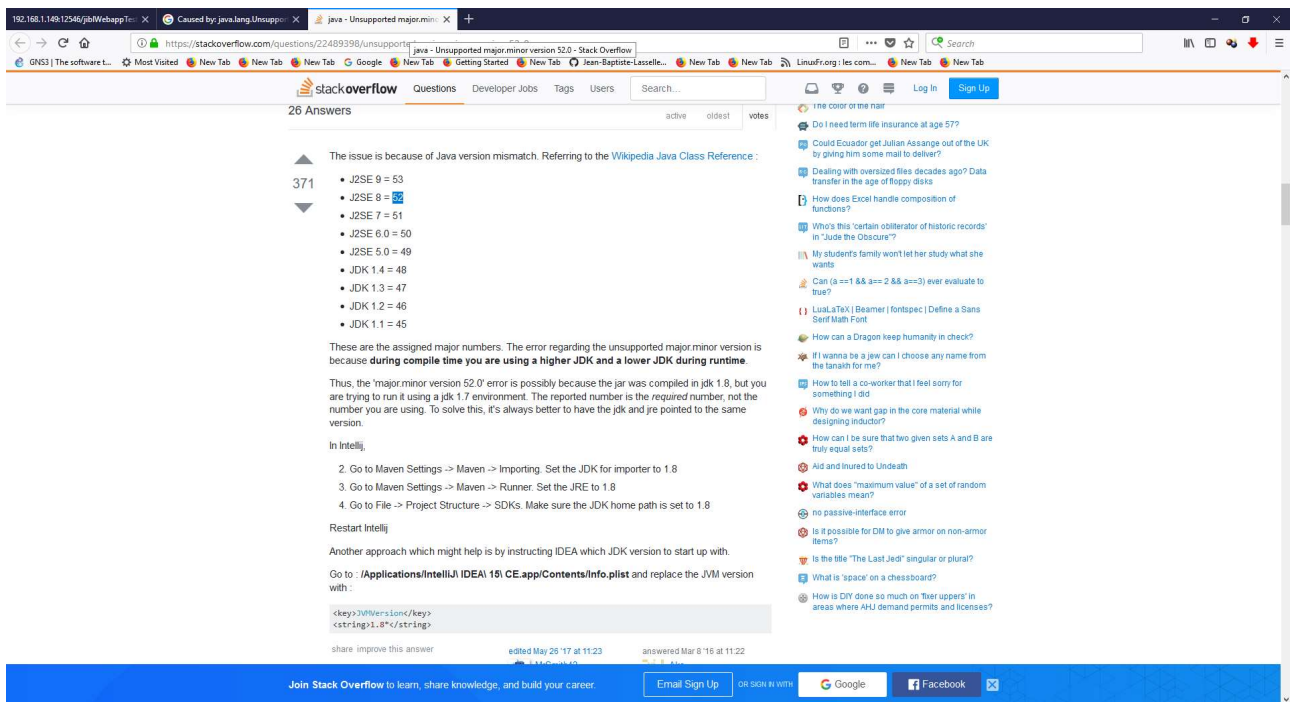
Donc j'ai quasiment finis d'automatiser le déploiement du datasource pour l'application  
L'erreur sur laquelle je tombe, indique entre autre une erreur de compatibilité entre la version de JRE,  
et le driver MariaDB/JDBC:

```
lauriane@lauriane-vm: ~  
.ContainerBase.addChildInternal ContainerBase.addChild: start:  
org.apache.catalina.LifecycleException: Failed to start component [StandardEngi  
ne[Catalina].StandardHost[localhost].StandardContext[/host-manager]]  
    at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:162)  
    at org.apache.catalina.core.ContainerBase.addChildInternal(ContainerBase  
.java:753)  
    at org.apache.catalina.core.ContainerBase.addChild(ContainerBase.java:72  
9)  
    at org.apache.catalina.core.StandardHost.addChild(StandardHost.java:717)  
    at org.apache.catalina.startup.HostConfig.deployDirectory(HostConfig.jav  
a:1126)  
    at org.apache.catalina.startup.HostConfig$DeployDirectory.run(HostConfig  
.java:1868)  
    at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:47  
3)  
    at java.util.concurrent.FutureTask.run(FutureTask.java:262)  
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.  
java:1152)  
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor  
.java:622)  
    at java.lang.Thread.run(Thread.java:748)  
Caused by: java.lang.UnsupportedClassVersionError: org/mariadb/jdbc/Driver : Uns  
upported major.minor version 52.0  
    at java.lang.ClassLoader.defineClass1(Native Method)  
    at java.lang.ClassLoader.defineClass(ClassLoader.java:803)  
    at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:14  
2)  
    at java.net.URLClassLoader.defineClass(URLClassLoader.java:442)  
    at java.net.URLClassLoader.access$100(URLClassLoader.java:64)  
lauriane@lauriane-vm:~$ sudo docker exec -it ciblededeploiement-composant-srv-jee /bin/bash -c "java --version"  
Unrecognized option: --version  
Error: Could not create the Java Virtual Machine.  
Error: A fatal exception has occurred. Program will exit.  
lauriane@lauriane-vm:~$ sudo docker exec -it ciblededeploiement-composant-srv-jee /bin/bash -c "java -version"  
java version "1.7.0_151"  
OpenJDK Runtime Environment (IcedTea 2.6.11) (7u151-2.6.11-2~deb8u1)  
OpenJDK 64-Bit Server VM (build 24.151-b01, mixed mode)  
lauriane@lauriane-vm:~$
```

Ci-dessus, ce sont les logs de tomcat dans le conteneur, et on voit une ligne :

"Caused by: java.lang.UnsupportedClassVersionError: org/maraidb/jdbc/Driver : Unsupported  
major.minor version 52.0"

Hors, le code 52 est associé au JDK 8 pour ce type d'exceptions natives Java, et elle est émise par la  
classe Pilote JDBC du pilote MariaDB/JDBC:



Bon, et de plus on voit que la version de java dans le conteneur qui exécute tomcat est:

```

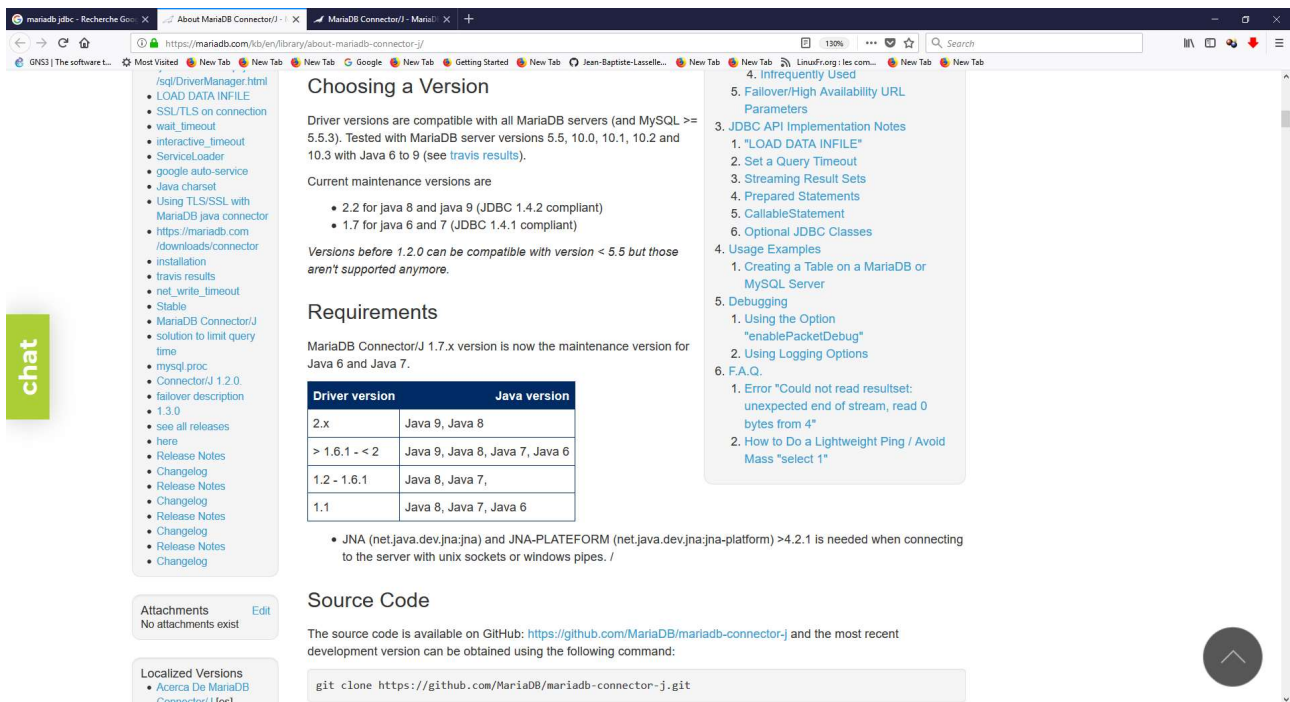
lauriane@lauriane-vm: ~
ContainerBase.addChildInternal ContainerBase.addChild: start:
org.apache.catalina.LifecycleException: Failed to start component [StandardEngine[Catalina].StandardHost[localhost].StandardContext[/host-manager]]
    at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:162)
    at org.apache.catalina.core.ContainerBase.addChildInternal(ContainerBase.java:753)
    at org.apache.catalina.core.ContainerBase.addChild(ContainerBase.java:729)
    at org.apache.catalina.core.StandardHost.addChild(StandardHost.java:717)
    at org.apache.catalina.startup.HostConfig.deployDirectory(HostConfig.java:1126)
    at org.apache.catalina.startup.HostConfig$DeployDirectory.run(HostConfig.java:1868)
    at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:473)
    at java.util.concurrent.FutureTask.run(FutureTask.java:262)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1152)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:622)
    at java.lang.Thread.run(Thread.java:748)
Caused by: java.lang.UnsupportedClassVersionError: org/mariadb/jdbc/Driver : Unsupported major.minor version 52.0
    at java.lang.ClassLoader.defineClass1(Native Method)
    at java.lang.ClassLoader.defineClass(ClassLoader.java:803)
    at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
    at java.net.URLClassLoader.defineClass(URLClassLoader.java:442)
    at java.net.URLClassLoader.access$100(URLClassLoader.java:64)
lauriane@lauriane-vm:~$ sudo docker exec -it ciblededeploiement-composant-srv-jee /bin/bash -c "java --version"
Unrecognized option: --version
Error: Could not create the Java Virtual Machine.
Error: A fatal exception has occurred. Program will exit.
lauriane@lauriane-vm:~$ sudo docker exec -it ciblededeploiement-composant-srv-jee /bin/bash -c "java -version"
java version "1.7.0_151"
OpenJDK Runtime Environment (IcedTea 2.6.11) (7u151-2.6.11-2~deb8u1)
OpenJDK 64-Bit Server VM (build 24.151-b01, mixed mode)
lauriane@lauriane-vm:~$

```

Donc il va falloir que je customise le dockerfile de la construction de tomcat. Dans ce dockerfile, je ferai l'installation de la JRE 8 au lieu de la JRE 7.

la doc officielle mariadb JDBC confirme l'incompatibilité (sachant que c'est la version 2.2.1 que je déployais pour le test) :





J'ai eut une autre déclinaison de mes problèmes d'incompatibilité avec la version de JRE dans le conteneur docker:

mes propres classes étaient compilées pour une cible JRE 8 (avec un source en Java 8), et ça re-levait une exception d'incompatibilité (et sans aucun log des exceptions de la webapp dans les logs serveurs):

J'ai testé ça en écrivant une simple Servlet faisant une réponse par une page HTML: en l'appelant directement avec son <url-pattern>, j'ai obtenu cette erreur d'incompatibilité entre la version de JRE, et la version de JDK que j'ai utilisé pour compiler mon code, avec les deux paramètres (source/target). Bref, j'ai changé la configuration du "maven-compiler-plugin", pour que mon code soit compilé en 1.7 en source, et en 1.7 en target, et le problème a été résolu, ma Servlet fonctionne (et permet de gérer les exceptions survenues dans l'application).

Arrivé là, j'ai modifié mon application pour qu'elle fasse un traitement des exceptions par config <error-page> dans le web.xml. Les exceptions sont attrapées par une Servlet, qui fait l'affichage d'autant d'infos que possible à propos de l'exception survenue. Une page jsp de test des exceptions permet de vérifier qu'une NullPointerException Levée, est bien traitée par ma servlet `{@see GestionnaireDexceptions }`. Tandis qu'aucune exception n'est attrapée lorsque j'invoque la page faisant usage de mon datasource configuré dans le web.xml

Oui, là, j'ai un problème de complexité pour mes tests:

version du driver jdbc / mariadb  
version de tomcat

version de dbcp  
version de Java dans le conteneur  
version de Java en source et target de compilation (pom.xml)  
fichier de configuration \$CATALINA\_BASE/conf/context.xml  
fichier de configuration \$CATALINA\_BASE/conf/server.xml  
fichier de configuration \$MONPROJET/WEB-INF/context.xml

Si j'ai seulement 2 versions de chaque, j'ai un gros nombre de combinaisons à tester"

Je vais traiter ce problème plus en profondeur, et en prenant en compte nos contraintes de projet en temps, à partir du tag: POINT\_DE\_RETOUR\_COMPLEXITE\_TESTS\_TROP\_LENTS

- **Pour Wildfly**
- **==>> il suffit d'utiliser jboss-cli.sh, comme on fait d'habitude, pour déployer le pilote jdbc, et faire l'opération /datasource-add etc...**
- **ccc**